Process to Connect Client to Server via Proxy for Cephalometric Analysis Program

W. Bholsithi, C. Sinthanayothin
National Electronic and Computer Technology Center, National Science and Technology Development Agency
112 Thailand Science Park, Phaholyothin Road
Klong 1, Klong Luang, Pathum Thani, 12120 Thailand

Abstract - Internet connection between client and server for running the web applications and processing the data between clients and servers can be done without passing through the proxy. However, the security issues and the access restriction by the institutes which handle the internet access on the client side has compelled the user to make a connection to the server via the proxy. This paper will show how to make a connection via proxy by applying Indy components for C++ Builder to enable the connection between client and server using the login, user password, and proxy address and proxy port number from the registry to along with setting the maximum connect time and maximum red time to initialize a connection via a proxy before sending the encoded data for the identification of the user to the server. After that, Indy components will check if the connection has become successful. If the connection has become successful, disconnect and clear input buffer of the Indy component and perform decoding and reading the data returned from the server to see whether the server will give the permission to run the program. The results show that this can be done with success.

I. INTRODUCTION

Fig. 1. Example of Internet Connection via Proxy Server [1]

Internet connections to allow the communications between clients and the servers can be done through LAN, air card or proxy servers to process the data through the web applications. However, the security issues on the growing demands for web applications as mentioned in [1] and the demands for data security in the hospitals have compelled the institute to require the clients to make the internet connections through the proxy server of the hospitals and medical precincts [2]. Furthermore, the proxy connections between content servers and end users with the intermediary trans-coding system will improve the system scalability to fit the particular competence of client devices [3], and the example of internet connection via proxy as shown in Fig. 1. This paper will focus on the internet connection and communication via the proxy for the user authentication on the client side using Indy10 internet components for C++ Builder compiler on the cephalometric analysis application with a brief introduction to web services.

Introduction to Web Services

Web services shown in Fig. 2 are self-contained modular applications to be published and invoked over the Internet with well-defined interfaces describing the services provided through the access by client applications.

Fig. 2. Web Service Architecture [4]

Web services have been designed to allow a loose coupling between client and server to eliminate the need for a specific platform or programming language some web server applications for server implementation has allowed multiple applications through Simple Object Access Protocol (SOAP) which has been in use since March 1998 to permit data exchanges in a decentralized and distributed environment using Extensible Markup Language (XML) to encode remote procedure calls and typically uses Hypertext Transfer Protocol (HTTP) as a communications protocol [5][6] with the interface described in a format the machine can process such as Web Services Description Language (WSDL) published by web service applications to be published on the server side or imported on the client side by using wizard or command-line utility with the SOAP Envelope structure shown in Fig. 3.

Fig. 3. Web Service Architecture [6]
SOAP has been superseded by the WEB API developments as a part of Web 2.0[7] which is a web service emphasizing on Representational State Transfer (REST) based communication with a defined set of HTTP request messages introduced by Roy Fielding in his dissertation in 2000 [8] which help to control the macro-interactions of the four Web components such as origin servers, gateways, proxies and clients by applying the constrains on the standard operations including GET, POST, PUT, DELETE on HTTP without imposing limitations on the individual participants as the way REST governs the proper behavior of participants. This web service development redefines the web service with two major types including the REST-compliant Web services with the service primary purpose is to manipulate XML representations of Web resources using a uniform set of “stateless” operations; and arbitrary Web services to be exposed with an arbitrary set of operations. [9] The representation of Representational State Transfer (REST) in different Web Services Description Language (WSDL) versions is as shown in Fig. 4.

![REST Structure for both WSDL V1.1 and WSDL V. 2.0](image)

**Indy Components for Web Service Applications**

Although there are several ways to implement the web application service, the researcher has chosen Indy internet components to implement a web service due to user friendly compiler of Borland C++ Builder 6. Internet Direct (Indy) is a collection of open source components for internet connections consisted of library of blocking sockets for reading and writing the data that supports clients, servers, TCP, UDP, raw sockets, as well as over 100 higher level protocols such as SMTP, POP3, NNTP, HTTP, and so on, available for C#, C++, Delphi, Visual Basic.NET, any .NET language, and Kylix and also applicable in Borland C++ Builder since Indy has been included as a set of components in Borland C++ Builder but users have to remove the old version and replace with the latest version (Indy10) in Borland C++ Builder.[11]

**CephSmileV2 Cephalometric Analysis Software**

CephSmile [12] is a cephalometric analysis program jointly developed by Faculty of Orthodontics, Dept. of Dentistry, Mahidol University. This cephalometric analysis program can perform lateral analysis, postero-anterior analysis, dental model analysis and superimposition on lateral cephalometric analysis for comparisons and 3D skull simulation from the trace lines and landmarks on the lateral cephalogram and the corresponding trace lines and landmarks on postero-anterior cephalogram. The latest version of CephSmile (CephSmileV2) shown as a diagram in Fig 5 has been developed for web services allowing content access into the database via the web server to be mentioned in details about internet connection implementations in the next section.

![Diagram for Content Access for CephSmileV2 Program](image)

**II. IMPLEMENTATIONS**

The web service implementation for CephSmileV2 program starts from encoding the data using the current date and the registry key for CephSmileV2, and then initialize Indy components, invoke the functions to establish the connection and check the connection whether it has been established successfully or otherwise. If connection is successful, disconnecting the internet and read the encoded data from the received web page and decode the data. Data to be used for web service implementation consisted of the date, random number, the program registry, user name, the password to access into server with some encoding and decoding schemes.

### A. CephSmile Web Services via LAN

The way to implement a web service for CephSmileV2 program with allow LAN connection as the way to enable CephSmile V2 to send the encoded message and receive the encoded response from the online database for data security purposes is to follow the procedure by applying the following Indy components to the CephSmileV2 Forms listed as follows:

1. **TIdSockInfo component** (IdSockInfo1) is to implement SOCKS protocol support for Indy clients by setting up Internet Protocol version to Internet Protocol version 4 for this case and the Host variables including the client website (CheckWeb) and port number of the corresponding client website (CLIENT_PORT) to the default value of 80 in (1) before assigning IdSockInfo1 to TransparentProxy element of TIdIOHandlerStack in (2).

   \[
   \text{IdSockInfo1->IPVersion = Id_IPv4;} \\
   \text{IdSockInfo1->Host = CheckWeb;} \\
   \text{IdSockInfo1->Port = CLIENT_PORT;} \\
   \text{IdIOHandlerStack1->TransparentProxy = IdSockInfo1}
   \]

2. **TIdIOHandlerStack component** (IdIOHandlerStack1) is to initialize the IOHandler framework of TIdTCPClient component by setting the maximum reading and connecting time, critical elements for asynchronous internet connection shown in (3) before assigning IdIOHandlerStack1 to IOHandler of TIdTCPClient component (IdTCPClient1) shown in (4) with CONNECT_TIME as the maximum connect
time and READ_TIME as the maximum read time to be assigned to IdTCPClient1 via IdIOHandlerStack1.

\[ \text{IdIOHandlerStack1} \rightarrow \text{ConnectTimeout} = \text{CONNECT_TIME}; \]
\[ \text{IdIOHandlerStack1} \rightarrow \text{ReadTimeout} = \text{READ_TIME}; \]  \hspace{1cm} (3)
\[ \text{IdTCPClient1} \rightarrow \text{IOHandler} = \text{IdIOHandlerStack1}; \]  \hspace{1cm} (4)

3. TIdTCPClient component (IdTCPClient1) for establishing TCP client connection by the TCP Client address (CheckWeb) and TCP Client port number (CLIENT_PORT) via IdSocksInfo1 along with TCP Client maximum connect time at CONNECT_TIME msec and TCP Client maximum read time at READ_TIME msec via IdIOHandlerStack1 before establishing the internet connection for IdTCPClient1 by provoking Connect() function as the way to provoke Open() event to allocate buffers and reset state properties used in TIOHandler class variables, and IdTCPClient1 calls Connected() to check whether the connection has been actually established in (5). If the TCP Client has connected successfully, call Disconnect() to end the connection before reading received data in (6).

\[ \text{IdTCPClient1} \rightarrow \text{Host} = \text{CheckWeb}; \]
\[ \text{IdTCPClient1} \rightarrow \text{Port} = \text{CLIENT_PORT}; \]
\[ \text{IdTCPClient1} \rightarrow \text{Connect}(); \]
\[ \text{bool test} = \text{IdTCPClient1} \rightarrow \text{Connected}(); \]
\[ \text{if (test == true)} \]
\[ \text{IdTCPClient1} \rightarrow \text{Disconnect}(); \ldots \]  \hspace{1cm} (5)

4. TIdHTTP component (IdHTTP1) is a component to read the information from the client website applied only after invoking Disconnect() event. Before using IdHTTP1 to read data, set the maximum time for reading the web page content at READ_TIME msec, and set content type of the request variables of IdHTTP1 to read encoded data before invoking Post() to read data assigned from Receive variable in (7) and call Seek(x,y) to read data started from x is zero and y is zero shown in (8) before loading the received data into TMemo component and assigning the text data in Memo1 to OutputFromWWW variable as shown in (9). The decoding scheme will be omitted.

\[ \text{IdHTTP1} \rightarrow \text{ReadTimeout} = \text{CONNECT_TIME1}; \]
\[ \text{IdHTTP1} \rightarrow \text{ReadTimeout} = \text{READ_TIME1}; \]
\[ \text{IdHTTP1} \rightarrow \text{Request} \rightarrow \text{ContentType} = \]
\[ \text{"application/x-www-form-urlencoded"}; \]
\[ \text{IdHTTP1} \rightarrow \text{Post} (\text{Trim(web)}, \text{data, Receive}); \]  \hspace{1cm} (7)
\[ \text{Receive} \rightarrow \text{Seek}(0, 0); \]  \hspace{1cm} (8)
\[ \text{Memo1} \rightarrow \text{Lines} \rightarrow \text{LoadFromStream} (\text{Receive}); \]
\[ \text{OutputFromWWW} = \text{Memo1} \rightarrow \text{Lines} \rightarrow \text{GetText}(); \]  \hspace{1cm} (9)

B. CephSmile Web Services via Air Card Connection

The way to implement a web service for CephSmileV2 program with the connection via air card connection is much the same way as the web service implementation via LAN connection with the differences at the maximum time for connecting and reading time have been set to -1 to allow indefinite maximum time for connection via Air Card for IdTCPClient1 in (10) and IdHTTP1 in (11).

\[ \text{IdTCPClient1} \rightarrow \text{ConnectTimeout} = -1; \]
\[ \text{IdTCPClient1} \rightarrow \text{ReadTimeout} = -1; \]  \hspace{1cm} (10)
\[ \text{IdHTTP1} \rightarrow \text{ConnectTimeout} = -1; \]
\[ \text{IdHTTP1} \rightarrow \text{ReadTimeout} = -1; \]  \hspace{1cm} (11)

From the empirical experiments, this method cannot be applied for the LAN connection cases since the LAN connection never allow the indefinite maximum connection time and maximum read time for both TIdTCPClient and TIdHTTP components.

C. CephSmile Web Services via Proxy Connection

The way to implement a web service for CephSmileV2 program with allow proxy connection is to add the following Indy components into the Forms of CephSmileV2:

1. TIdSocksInfo component (IdSocksInfo1) is to implement SOCKS protocol support for Indy clients by setting up Internet Protocol version to Internet Protocol version 4 for this case, and the initializations of variables relating to proxy connection for TIdHTTP component such as proxy server (ProxyServer), proxy port (ProxyPort), user name (UserName) and proxy password (LogInPassword) before assigning IdSocksInfo1 to TransparentProxy of TIdSSLSIOHandlerSocketOpenSSL component (IdIOHdSockSSL1) in (12).

\[ \text{IdSocksInfo1} \rightarrow \text{IPVersion} = \text{Id_IPv4}; \]
\[ \text{IdSocksInfo1} \rightarrow \text{Host} = \text{ProxyServer}; \]
\[ \text{IdSocksInfo1} \rightarrow \text{Port} = \text{StrToInt(ProxyPort)}; \]
\[ \text{IdSocksInfo1} \rightarrow \text{Username} = \text{UserName}; \]
\[ \text{IdSocksInfo1} \rightarrow \text{Password} = \text{LogInPassword}; \]
\[ \text{IdIOHdSockSSL1} \rightarrow \text{TransparentProxy} = \text{IdSocksInfo1} \]  \hspace{1cm} (12)

2. TIdSSLSIOHandlerSocketOpenSSL component (IdIOHdSockSSL1) is a component to create SSL (Secure Socket Layer) and TLS (Transport Layer Security) IOHandler to manage encrypted connection by proxy via OpenSSL Shared Library files such as libeay32.dll [19], an open source DLL with encryption functions to allow coded communication through networks and ssleay32.dll [20], the DLL file to access into The OpenSSL Toolkit functions in The OpenSSL Project to implement the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols and a full-strength general purpose cryptography library [21][22] by setting IdIOHdlerSockSSL1 to show exception when it has
reached the maximum line of code, set the SSL protocol to SSL v2/v3, set the SSL protocol for the client, set verification mode to to force checking if the other side has a proper valid certificate with the verification depth to MaxLevels layers, and set OnGetPassword to control password obtaining procedure before assigning IdIOHanderSockSSL1 to IOHandler of IdHTTP1 in (13) and functions in OnGetPassword of IdHTTP1 in (14).

idIOHdSockSSL1->MaxLineAction = maException;
idIOHdSockSSL1->SSLOptions->Method = sslvSSLv23;
idIOHdSockSSL1->SSLOptions->Mode = sslmClient;
idIOHdSockSSL1->SSLOptions->VerifyDepth = MaxLevels;
idIOHdSockSSL1->SSLOptions->VerifyMode = sslvrfPeer;
idIOHdSockSSL1->OnGetPassword = IdIOHdSockSSL1GetPassword;
idHTTP1->IOHandler = IdIOHdSockSSL1; (13)
if(AdvOfficePager1->ActivePage == AdvOfficePage3) {
  if(ProxyPasswordEdit->Text != "")
    Password = ProxyPasswordEdit->Text; (14)
}
}

3. TIdHTTP component (IdHTTP1) to initialize proxy connection by setting up the maximum times for attempted connections to a certain level (MaxRetry), enable the proxy redirect by setting HandleRedirect of IdHTTP1 to true, set the maximum connect time to CONNECT_TIME1 msec and the maximum read time to READ_TIME1 msec while keeping the proxy connection to alive and set the response to keep connection alive. Furthermore, it is necessary to clear the proxy parameters by calling Clear() before setting the values of proxy parameters including the proxy user name, proxy password, proxy server, proxy server port number in (15). Call Get() to obtain data from ConnectingWeb string created by adding “http://” string to CheckWeb which is the server website to connect with server before calling Connected() function as the way to check whether the connection has been successfully created in (16). For the successful connection case, call disconnect function in (17) and clear input buffer in IOHandler of IdHTTP1 to prevent EidConnClosedGracefully error that shuts down the internet connection with no order in (18), allocate the memory for ReceivePage to get the data from the server in (19), set the maximum time for reading the web page content at READ_TIME1 msec, and set content type for the request variable in IdHTTP1 to read encoded data before invoking Post() to read data from Receive variable in (20) and call Seek(x,y) event to read data started from x is zero and y is zero in (21) before loading the received data to TMemo and assigning the text data in Memo1 to OutputFromWWW in (22) while omitting the decoding function.

AnsiString ProxyPortStr = StrToInt(ProxyPort);
idHTTP1->MaxAuthRetries = MaxRetry;
idHTTP1->HandleRedirects = true;
idHTTP1->ConnectTimeout = CONNECT_TIME1;
idHTTP1->ReadTimeout = READ_TIME1;
idHTTP1->ProxyParams->Clear();
idHTTP1->ProxyParams->ProxyServer = ProxyServer;
idHTTP1->ProxyParams->ProxyPort = ProxyPortStr;
idHTTP1->ProxyParams->ProxyUsername = UserName;
idHTTP1->ProxyParams->ProxyPassword = LoginPassword;
idHTTP1->Request->Connection = "Keep-Alive";
idHTTP1->Request->ProxyConnection = "Keep-Alive";
idHTTP1->Response->KeepAlive = true; (15)

AnsiString CheckWeb = web;
AnsiString ConnectingWeb = "http://" + CheckWeb;
idHTTP1->Get(ConnectingWeb, Receive);
bool test = IdHTTP1->Connected(); (16)
if (test == true) {
  IdHTTP1->Disconnect(); (17)
  if(IdHTTP1->IOHandler != NULL) {
    IdHTTP1->IOHandler->InputBuffer->Clear(); (18)
  }
  TMemoryStream *RPage = new TmemoryStream; (19)
  IdHTTP1->Request->ContentType = "application/x-www-form-urlencoded";
idHTTP1->ReadTimeout = READ_TIME1;
idHTTP1->Post(Trim(web), data, RPage); (20)
  ReceivePage->Seek(0, 0); (21)
  Memo1->Clear();
  Memo1->Lines->LoadFromStream(ReceivePage);
  OutputFromWWW = Memo1->Lines->GetText(); .... (22)
}

4. It is necessary to invoke OnAuthorization event of TIdHTTP component to control the proxy server connection by assigning the proper user name and password to the Authentication variable and setting a proper logic to the Handle variable if ActivePage is AdvOfficePage3, the Proxy connection page in (23).

if(AdvOfficePager1->ActivePage == AdvOfficePage3) {
  if(UserNameEdit3->Text != "")
    Authentication->Username = UserNameEdit3->Text;
  if(PasswordEdit3->Text != "")
    Authentication->Password = PasswordEdit3->Text; (23)
}
5. Another TidHTTP event to be invoked is OnProxyAuthorization event to set up the proper user name and password to the Authentication variable for proxy server and set proper logic to the Handle the authorization via proxy variable in (24).

```cpp
if(AdvOfficePager1->ActivePage == AdvOfficePage3)
{
  if(ProxyLogInEdit->Text != "")
  {
    Authentication->Username = ProxyLogInEdit->Text;
  }
  if(ProxyPasswordEdit->Text != "")
  {
    Authentication->Password = ProxyPasswordEdit->Text;
  }
}
```

6. The TidHTTP OnSelectAuthorization event will be invoked by selecting a proper AuthenticationClass if ActivePage is the page for Proxy connection in (25).

```cpp
if(AdvOfficePager1->ActivePage == AdvOfficePage3)
{
  AuthenticationClass = __classid(TIdNTLMAuthentication);
}
```

7. The last event of TidHTTP to be invoked is OnSelectProxyAuthorization event to select the proper AuthenticationClass for the connection via proxy server as shown in (26).

```cpp
if(AdvOfficePager1->ActivePage == AdvOfficePage3)
{
  if (AnsiPos("Proxy-Authenticate: NTLM", IdHTTP1->Response->RawHeaders->Text) >= 1)
  {
    IdHTTP1->ProxyParams->Clear();
    IdHTTP1->ProxyParams->BasicAuthentication = false;
    AuthenticationClass = __classid(TIdNTLMAuthentication);
  }
  else if (AnsiPos("Proxy-Authenticate: Basic", IdHTTP1->Response->RawHeaders->Text) >= 1)
  {
    AuthenticationClass = __classid(TIdBasicAuthentication);
    IdHTTP1->ProxyParams->BasicAuthentication = true;
  }
  else if (AnsiPos("Proxy-Authenticate: Digest", IdHTTP1->Response->RawHeaders->Text) >= 1)
  {
    AuthenticationClass = __classid(TIdDigestAuthentication);
    IdHTTP1->ProxyParams->BasicAuthentication = true;
  }
}
```

III. EXPERIMENTS AND RESULTS

The user has applied the HKEY_CURRENT_USER root to track down the registry to find ProxyServer value containing the proxy server address according to the codes in (27) showing only the case of finding the proxy server while omitting the rest. Cephsmile program before starting the connection via proxy server shown in Fig. 6(A) and the result after filing the log in along with passwords to both server and proxy to start a web service via proxy shown in Fig.6(B).
IV. FURTHER WORKS AND CONCLUSIONS

Although the web services via proxy server connection has become successful, the user wishes to modify ReadTimeout and ConnectTimeout of both TIdTCPClient and TIdHTTP to allow flexible maximum reading and connecting times for the different types of internet connections instead of relying on the inconvenient trial and error methods to set up the maximum reading and connecting times. Further developments have to take IPv6 IP addresses into account as the IP address registration has been changed as the example in [23] has implied. The measures to deal with malwares exploiting the Internet protocol are in need as the case of worms exploiting IPv6 and IPv4-IPv6 dual-stack networks in [24] has suggested.

ACKNOWLEDGMENT

We appreciate the contributions from the orthodontist team from the Department of Orthodontics, Faculty of Dentistry, Mahidol University for raising the issue of web services via proxy server connection and granting a permission to work at the faculty to solve the problems along with special thanks to Dr. Tuveesup Apiwattanaapong and Mr. Ekekachan Rattanaleadnusorn from Software Engineering Lab (SWE) to help us on the issue of applying TIdHTTP component for connections via a proxy and special thanks to IMS, Medical Cluster, National Electronics and Computer Technology Center (NECTEC), and NSTDA for grant support.

REFERENCES